MISB MOTION IMAGERY STANDARDS BOARD	MISB ST 1303.1
STANDARD **Multi-Dimensional Array Pack**	**1 November 2018**

1 Scope

This standard describes a method for formatting multi-dimensional arrays of data in KLV (Key Length Value). A multi-dimensional array is used to store and organize related data; typically, the array is processed as a unit of data (i.e. matrixes, etc.). This standard provides a Pack construct to format multi-dimensional arrays, which minimizes the bytes needed to represent the data. In addition, this standard includes options for specifying predefined methods of array element processing, such as using MISB ST 1201 (Floating Point to Integer Mapping) for compression.

In application, the Multi-Dimensional Array Pack defined in this standard requires further context from an invoking document; that is, it is not used standalone. To provide consistency, this standard defines a syntax for invoking this standard within MISB documents.

2 References

[1] SMPTE ST 336:2017 Data Encoding Protocol Using Key-Length-Value.

[2] MISB ST 1201.3 Floating Point to Integer Mapping, Oct 2017.

[3] MISB MISP-2019.1: Motion Imagery Handbook, Nov 2018.

[4] MISB ST 0807.22 MISB KLV Metadata Registry, Jun 2018.

[5] ISO/IEC 8825-1:2015 (ITU-T X.690) Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canocical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[6] IEEE 754-2008 Standard for Floating-Point Arithmetic.

3 Modifications and Changes

Revision	Date	Summary of Changes
ST 1303.1	11/01/2018	• Modified Section 8.2 Example 3 invoking statement dimension lengths to match the number of dimensions • Deprecated Req -06 as it is a definition • Renamed Appendix A to be Deprecated Requirements & re-ordered the appendices to their occurrence as referenced

		• Updated References; deleted reference 0701; Added reference MISP Motion Imagery Handbook [3] • Editorial cleanup across the entire document: consistent usage of terms, registry vs dictionary and grammatical usage

4 Acronyms

IMAPA	Integer Mapping Starting Point A; defined in MISB ST 1201
IMAPB	Integer Mapping Starting Point B; defined in MISB ST 1201
KLV	Key Length Value
MISB	Motion Imagery Standards Board
MISP	Motion Imagery Standards Profile
SMPTE	Society of Motion Pictures & Television Engineers
ST	Standard

5 Introduction

Within software, data structures such as records and arrays provide information organization. When reformatting data structures into KLV record structures map readily into KLV constructs, such as KLV Set and Pack constructs. However, the KLV standard [1] does not formally specify a means for reformatting arrays into KLV. This standard defines an efficient and flexible method for formatting multi-dimensional arrays of fixed length into KLV.

6 Array Composition

An array of data is an organization of multiple fixed-sized elements (i.e. the data) along one or more dimensions. In transferring an array of data, support information ensures the organization of the data is consistent between the sender and receiver. A document which invokes this standard will define the array parameters to include: the number of dimensions, the number of elements per dimension, the size of an element, and the linkage of the elements to its defining KLV registry.

The elements in an array can be of any type (e.g. integers, floating point values, strings, KLV pack or set values, etc.), however, all elements need to be the same number of bytes. Elements of different sizes are normalized to the same length by either inflating or shrinking (see Appendix B). When grouping data into an array there is an opportunity to use the information about the group to reduce the number of bytes needed for the array. For example, if an array is composed of integers originally defined with four bytes each, yet all the values only need one byte then the array uses one-byte elements instead of four.

In this document the following terminology is used:

Array: Uppercase notation to indicate reference to an array as defined in this document.

Element: Data representing a value identified by an index or indices.

Requirement(s)	
ST 1303-01	All elements in an Array shall use the same number of bytes.
ST 1303-02	The invoking standard shall specify the number of dimensions of the Array.
ST 1303-03	The invoking standard shall specify the size of each dimension or how to compute the size of each dimension in the Array.
ST 1303-04	The invoking standard shall specify the data contained in the Array using KLV Keys or KLV symbols.

Figure 1 illustrates a single dimension Array, where its Elements are indexed beginning at 0 and ends at N-1 for N Element columns. Elements in a one-dimensional array index only by columns, so column 0 is Element 0, column 1 is Element 1, etc.

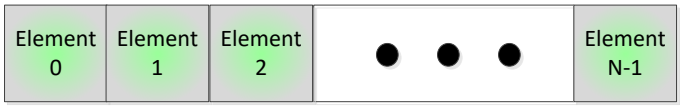

Figure 1: Illustration of a One-Dimensional Array of N Elements.

Figure 2 shows a two-dimensional Array, where its Elements index by the row r and the column c. Thus, the first Element is (0, 0) for r = 0, c = 0; the second Element in row 0 is (0, 1) for r = 0, c = 1; the second Element in row 1 is (1, 1) for r = 1 and c = 1, etc. (see Figure 2). Rows are indexed 0 to M-1 and columns are indexed 0 to N-1.

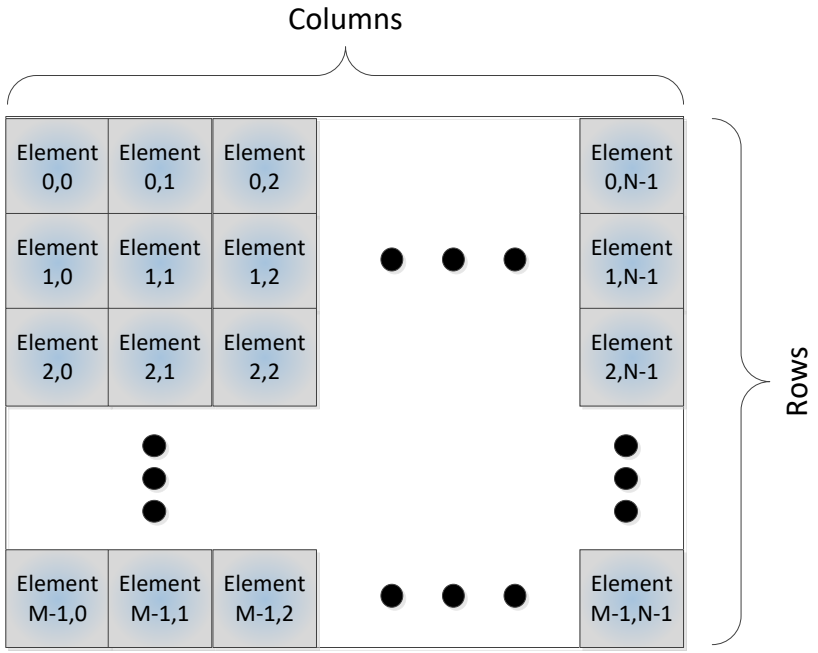

Figure 2: Illustration of a Two-Dimensional Array of M-Rows by N-Columns

Figure 3 depicts a three-dimensional Array, where its Elements are indexed by planes, rows and columns. Each plane is shown as a separate colored two-dimensional array in Figure 3. Again, the first Element in plane 0, row, 0 and column 0 is (0, 0 ,0) for (p, r, c) where p = r = c = 0. Planes are indexed 0 to P-1; rows are indexed 0 to M-1, and columns are indexed 0 to N-1.

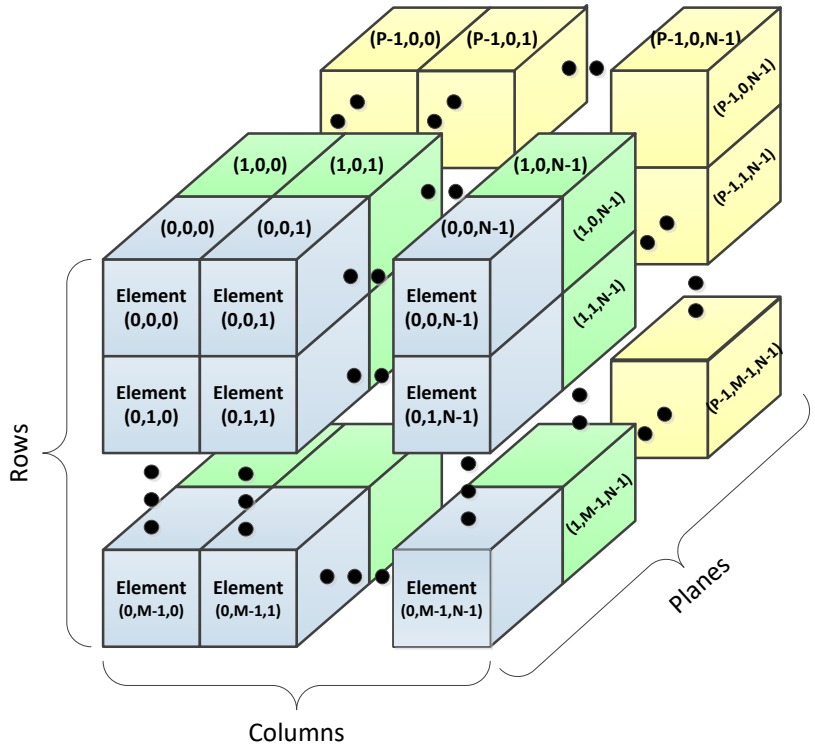

Figure 3: Illustration of a Three-Dimensional Array of P-Planes by M-Rows by N-Columns

Array have two types of Elements: Registry and Processed. Registry Elements match the type and bit structure as defined in their KLV registry. Processed Elements are manipulated before they are inserted into the Array. The manipulation is typically a method of compression; such an example is using MISB ST 1201 [2] (see **Error! Reference source not found.**) to compress floating point values before they are inserted into the Array. Processed elements require additional support information to specify the processing method along with any parameters needed to do the processing. Each processing method, called an EPA (Element Processing Algorithm), is assigned a value, which is documented in Table 3 of Appendix D.

7 Multi-Dimensional Array Pack

To provide the most bit-efficient method for transmitting the Array, plus its required support information, and its optional EPA information, a KLV Truncation Pack (described in the Motion Imagery Handbook [3]) is used. The parameters of the Multi-Dimensional Array Pack are in Table 1.

Table 1: Multi-Dimensional Array Pack

Name	Required/ Optional	Type	Min Value	Description
N_{Dim}	Required	BER-OID Subidentifier Integer	1	Number of dimensions in the Array
Dim_i	Required	BER-OID Subidentifier Integer	1	Size of i^{th} dimension - There are N_{Dim} number of these values. There is always at least one dimension, so this value will always be defined.
E_{Bytes}	Required	BER-OID Subidentifier Integer	1	Number of bytes for each element - the minimum value for this number is zero.
EPA	Required	BER-OID Subidentifier Integer	1	Element Processing Algorithm. Indicator of Element Processing Algorithm to perform on each array element.
EPAS	Optional	Defined by EPA	N/A	Element Processing Algorithm Support
Array of Elements	Optional	Defined by Invoking Document	N/A	Array of data serialized into a one-dimensional array using row major ordering

Requirement	
ST 1303-05	A Multi-Dimensional Array Pack shall have its mandatory parameters N_{DIM}, Dim_i, E_{Bytes}, EPA and its optional elements ordered as defined in MISB ST 1303 Table 1: Multi-Dimensional Array Pack.

Figure 4 illustrates the Multi-Dimensional Array Pack, along with its Key and Length. The parameters (blue) are the required information stated in Table 1; EPAS parameters (yellow) are optional; the Array of Elements (green) constitutes the Array data, which is also optional.

Key	Length	Multi-Dimensional Array Pack "Value"								
		N_{Dim} (BER-OID)	Dim_1 (BER-OID)	Dim_2 (BER-OID)	...	Dim_N (BER-OID)	E_{Bytes} (BER-OID)	EPA (BER-OID)	EPAS (Variable)	ARRAY of Elements (Dim_1*Dim_2...*Dim_N*E_{Bytes})

Figure 4: Multi-Dimensional Pack Structure

The following sections describe the KLV components of the Multi-Dimensional Array Pack.

7.1 Multi-Dimensional Array Pack – Key

The Multi-Dimensional Array Pack 16-Byte UL[1] "Key" is registered in MISB ST 0807[4] as:

06.0E.2B.34.02.05.01.01.0E.01.03.03.06.00.00.00 (CRC 39697)

7.2 Multi-Dimensional Array Pack – Length

The Length of the Multi-Dimensional Array Pack is BER-length encoded as specified in [1]. The length includes the number of bytes for N_{Dim}, all the dimensions ($Dim_1 \ldots Dim_N$), the number of bytes for each element E_{Bytes}, the EPA, the EPAS, and the Array of Elements. Compute $Pack_{Length}$ as follows:

$Pack_{Length} = L(N_{Dim}) + L(Dim_1) + \ldots + L(Dim_N) + L(E_{Bytes}) + L(EPA) + L(EPAS) + L(Array\ of\ Elements)$

Where L(**x**) is the length of value **x** in bytes.

As discussed below, the L(EPAS) and L(Array) can both be zero depending on usage and circumstances. See Appendix D on interpreting the EPAS if included.

7.3 Multi-Dimensional Array Pack – Value

The Value of the Multi-Dimensional Array Pack consists of a collection of required and optional parameters as listed in the following sections. All parameters are required unless noted.

7.3.1 Parameter - N_{Dim}

The N_{Dim} parameter is the count of dimensions in the Array of Elements. For example, a simple list of Elements is a one-dimensional Array; therefore, N_{Dim} equals 1. A rectangle of Elements is a two-dimensional Array; therefore, N_{Dim} equals 2. A cube of elements is a three-dimensional Array; therefore, N_{Dim} equals 3, etc. A value of zero for N_{Dim} is not allowed.

Requirement(s)	
ST 1303-07	The invoking document shall specify a value for N_{Dim} that is greater than or equal to one (1).
ST 1303-08	The encoding of N_{Dim} shall be BER-OID [5].

Because N_{Dim} is BER-OID [5] encoded the number of bytes is based on the value of the parameter. For example, for a parameter value less than 127 dimensions one byte is used and $L(N_{Dim}) = 1$. The value of the N_{Dim} parameter specifies the number N of dimensional parameters ($Dim_1 \ldots Dim_N$) following the N_{Dim} parameter. For example, if $N_{Dim} = 3$ (a cube of Elements) then Dim_1, Dim_2 and Dim_3 parameters will follow the N_{Dim} parameter.

[1] Note: this UL is only used within an invoking document and not standalone

7.3.2 Parameter – Dim_i

The Dim_i parameter specifies the number of elements in the i^{th} dimension, which is greater than or equal to one (i.e. zero is not allowed). For example, for a three-dimensional cube Array of 100x200x300 Elements (i.e. $N_{Dim} = 3$) $Dim_1 = 100$, $Dim_2 = 200$ and $Dim_3 = 300$.

Requirement(s)	
ST 1303-09	The value of Dim_i shall be greater than or equal to one (1).
ST 1303-10	The encoding of each Dim_i shall be BER-OID [5].

Because each of these parameters is BER-OID encoded the number of bytes used for each parameter is based on its value. In the example above $L(Dim_1) = 1$ byte (since 110 <127), $L(Dim_2) = 2$ bytes (since 200 > 127), and $L(Dim_3) = 2$ bytes (since 300 > 127).

7.3.3 Parameter – E_{Bytes}

The E_{Bytes} parameter specifies the number of bytes representing each Element within the Array. For example, if the Array consists of single-precision, floating-point numbers, then $E_{Bytes} = 4$ bytes. Zero is a valid value for E_{Bytes}. E_{Bytes} equal to zero (0) indicates each Element in the Array is of zero length, and therefore, the Array of Elements parameter contains no data. A zero valued E_{Bytes} signals information without passing any data; for example, indicating there was no change in the last set of measurements.

Requirement(s)	
ST 1303-11	The encoding of E_{Bytes} shall be BER-OID [5].
ST 1303-12	The value of E_{Bytes} shall be greater than or equal to zero (0).
ST 1303-13	E_{Bytes} equal to zero (0) shall signal that the Array of Elements is not included in the Multi-Dimensional Array Pack.

Because E_{Bytes} is BER-OID encoded the number of bytes is based on the value of the parameter. For example, if $E_{Bytes} = 4$ bytes then only one byte is used i.e. $L(E_{Bytes}) = L(4) = 1$. Strings longer than 127 bytes will use more than one byte i.e. L(130)=2; however, all standard numerical values (i.e. Integers, Unsigned Integer, Floats) will usually be in the range of 1 to 8 bytes. The computation of the Array Length uses the E_{Bytes} value along with the values of each Dim_i.

7.3.4 Parameter – EPA (Element Processing Algorithm)

The EPA (Element Processing Algorithm) parameter specifies the method of processing when forming the Array of Elements, and the method for interpreting the Element values when parsing data from the Array. An EPA of one (0x01) indicates no processing, and the Elements match the definitions as specified in their KLV registry; all other EPA values indicate Elements have been processed. Appendix D lists the different Element Processing Algorithms and their parameters.

Requirement	
ST 1303-14	An EPA (Element Processing Algorithm) parameter value shall be assigned a value only from MISB ST 1303 Table 3: Element Processing Algorithms.

7.3.5 Parameters – EPAS (Element Processing Algorithm Support) (Optional)

The optional EPAS (Element Processing Algorithm Support) parameters specify values for an Element Processing Algorithm if needed. Each EPA in Appendix D lists its support values. If the EPA value is set to one (0x01) then there is no corresponding EPAS parameter. EPAS parameters come before the Array so parsers can use the information to interpret the data on input.

7.3.6 Parameter – Array of Elements (Optional)

The optional Array of Elements parameter is the multi-dimensional data being described by all required and optional parameters. The Array is a serialized block of data that contains Dim_1*Dim_2*…*Dim_N Elements, with each Element exactly E_{Bytes} bytes. If E_{Bytes} is zero the Array is considered empty (e.g. all zeros), and it is truncated from the Multi-Dimensional Array Pack.

The total length of the Array of Elements in bytes is: $Dim_1 * Dim_2 *\ldots * Dim_N * E_{Bytes}$.

To enable accessing the elements within the block of data, the Array is serialized in row major order.

Requirement	
ST 1303-15	The data in the Array of Elements shall be organized in row major order.

Because the block of data is organized in row major order, any Element in the Array can be accessed by computing an offset from the start of the Array (see Appendix B for computation).

As is done in many software languages (such as C, C++ or Java) elements in an array are indexed using zero-based notations, so the first element of each dimension is indexed as the $zero_{th}$ element. For example, Array(10,0) is the first column Element of the 11^{th} row for Array(r, c) where r = 10 and c= 0.

8 MISP and MISB Document Standard Notation

When using or "invoking" the Multi-Dimensional Array Pack it is important to be clear as to the type, size and optional packing used. The following notation will ensure consistency and completeness of the Array definition:

MDARRAY(<Data Identifier>,<N_{Dims}>, <Dim_1>,…,<Dim_N>, [E_{Bytes}])

Where:

Parameters in <..> are required.
Parameters in [..] are optional.
<Data Identifier> is the KLV Registry identifier(s) to use for the Elements (see Section 8.1).
Other parameters are defined in Section 7.

There are cases when some of the values in the notation will be unknown as an invoking MISB document is written (i.e. values computed at runtime); those values are to be labeled with a footnote (i.e. "Note$_x$") which references text providing further explanation. Footnotes can be used for multiple parameters and multiple uses of the MDARRAY definition - See Examples in Section 8.2. E_{Bytes} can be determined at runtime based on the data values in the Array; however, an invoking standard may specify a predetermined size. Although E_{Bytes} is a mandatory parameter in the Multi-Dimensional Array Pack, the Standard Notation lists it as "optional" because the E_{bytes} value may be determined at run time or from a known fixed length type (e.g. the registry lists the value as a 32-bit IEEE floating point number).

Requirement	
ST 1303-16	When using MISB ST 1303, the invoking standard shall use the MDARRAY(…) notation to ensure consistency and completeness of the Multi-Dimensional Array Pack definition in accordance with MISB ST 1303.

8.1 Data Identifier

The Data Identifier links the Array of Elements to a specific KLV registry item. The KLV registry defines the details of the Element, such as the type, name, description, etc. The Data Identifier can be either the Key or the Symbol Name. The Key is the 16-byte Universal Label associated with any KLV data item. The Symbol Name is the unique text identifier assigned to a KLV data item.

Data Identifiers can be associated to the whole Array (homogeneous), or to parts of the Array (heterogeneous) if the elements all share the same length in bytes. A heterogeneous example is a grid of points with values representing latitude, longitude and height above ellipsoid (HAE); a three-dimensional Array of points with a first plane used for latitude values, a second plane used for longitude values and a third plane for HAE values as illustrated in Figure 5. This example assumes the latitude, longitude and HAE values use the same number of bytes, e.g. all single-precision, floating-point values.

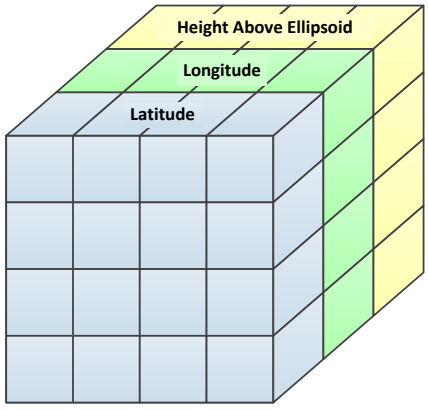

Figure 5: Illustration of 3-D Array with Different Plane Types

When the Array is heterogeneous additional footnotes describe the meaning and usage of the dimensions. The footnotes describe how the different dimensions link to the KLV registry. For example, in Figure 5, Array(0, all rows, all columns) links to the Latitude KLV item; Array(1, all rows, all columns) links to the Longitude KLV item, and Array(2, all rows, all columns) links to the HAE KLV item.

When the Array is heterogeneous and an EPA other than direct mapping is used, the Element values need to be compatible with the EPA Care must be taken in choosing the EPAS parameters because the data may have different value ranges. Example 1 and 2 below pertain to homogeneous Data Identifiers Arrays, while Examples 3, 4, and 5 show three different approaches to defining a heterogeneous Array.

8.2 Examples

Example 1: An Array of homogeneous Range Depth data.

MDARRAY(06.0E.2B.34.01.01.01.01.0E.01.01.03.25.00.00.00, 2, 100, 100)

or,

MDARRAY(range_depth, 2, 100, 100)

This is a two-dimensional Array of 100x100 Elements of Range Depth (UL specified) data.

Example 2: An Array of homogeneous Range Depth data with footnotes describing the "run-time" values.

MDARRAY(06.0E.2B.34.01.01.01.01.0E.01.01.03.25.00.00.00, 2, $Note_A$, $Note_A$)

$Note_A$: This value is dependent on the dimensions of the sensor. Implementations should use EPA 2, where Min and Max are the bounds of the Array data and the Element size (E_{Bytes}) is determined by IMAPA(Min, Max, 1.0e-4).

This is a two-dimensional Array of Range Depth data from a sensor (dimensions known only at runtime - $Note_A$). The invoker recommends the use of an EPA transformation based on the expected values in the Array. In this example, the recommendation is to use EPA two (2), so each Element's value maps to an integer with IMAPB by using the min/max of the source array data and E_{Bytes}. The recommended Element Size (E_{Bytes}) is computed based on the IMAPA computation for length using a precision value of 1.0e-4.

Example 3: An Array of heterogeneous Latitude/Longitude/Altitude data with footnotes describing the dimensional use and "run-time" values.

MDARRAY($Note_A$, 3, 3, $Note_B$, $Note_B$)

$Note_A$: Array(0, r, c) = 06.0E.2B.34.01.01.01.01.07.01.02.01.02.04.00.00[a],
Array(1, r, c) = 06.0E.2B.34.01.01.01.01.07.01.02.01.02.06.00.00[a],
Array(2, r, c) = 06.0E.2B.34.01.01.01.01.07.01.02.01.02.02.00.00[a],
r = all rows and c = all columns.

$Note_B$: This value is dependent on the dimensions of the sensor.

[a] KLV Universal Label (Key), taken from the SMPTE dictionary, illustrate the use of 4-byte floats; other KLV Keys in the DoD dictionary (MISB ST 0807) should be used for normal MISB applications.

This is a three-dimensional Array of Latitude, Longitude and Altitude data from a sensor (dimensions known only at runtime - $Note_B$). The first plane (plane 0) is the Latitude, the second plane (plane 1) is the Longitude and the third plane (plane 2) is the Altitude.

This usage is not recommended when applying EPA 2 (ST 1201 Floating Point to Integer Mapping) to reduce the Element size (E_{Bytes}) because it is impeded by the need to represent values with ranges orders of magnitude apart using a single mapping (latitudes of +/-90° versus HAE of -900 to 19000 meters).

Example 4: The same data of Example 3 but defined as three separate homogeneous Arrays.

MDARRAY(06.0E.2B.34.01.01.01.01.07.01.02.01.02.04.00.00[a], 2, $Note_A$, $Note_A$)
MDARRAY(06.0E.2B.34.01.01.01.01.07.01.02.01.02.06.00.00[a], 2, $Note_A$, $Note_A$)
MDARRAY(06.0E.2B.34.01.01.01.01.07.01.02.01.02.02.00.00[a], 2, $Note_A$, $Note_A$)

$Note_A$: This value is dependent on the size of the sensor.

[a] KLVUniversal Label (Key), taken from the SMPTE dictionary, illustrate the use of 4 byte floats; other KLV Keys in the DoD dictionary (MISB ST 0807) should be used for normal MISB applications.

These are three separate Arrays: the first Array contains Latitude values; the second Array contains Longitude values; the third Array contains Altitude values. The dimension of each Array is defined at runtime from a sensor's information (dimensions known only at runtime - $Note_A$).

This usage is preferable when applying EPA 2 (MISB ST 1201 Floating Point to Integer Mapping) to reduce the Element size (E_{Bytes}) as each Array of Elements transforms individually.

Example 5: An ARRAY using the Location Truncation Pack.

MDARRAY(location_pack, 2, rows, cols)

This is a two-dimensional Array of location_pack values, with each value potentially defining sub-components: latitude, longitude, height, sigma latitude, sigma longitude, sigma height, Rho lat/lon, Rho lat/height and Rho lon/height. The Element size (E_{Bytes}) determines which data is included in each Elements pack - the Elements must be consistently defined to maintain the required fixed Element size.

This usage rigidly defines the ranges of each sub-component based on the pack definition, so the advantage of adjusting the Element size (E_{Bytes}) based on the range of data within the Array cannot be used.

Appendix A Deprecated Requirements

Requirement	
ST 1303-06 (Deprecated)	The 16-byte Key for the Multidimensional Array Truncation Pack shall be 06.0E.2B.34.02.05.01.01.0E.01.03.03.06.00.00.00 (CRC 39697).

Appendix B Row Major Computation

The following equations demonstrate how to compute the offsets into an Array based on the Array dimensions, D_1, $D_2 \ldots D_N$ and the desired Element indexes x_1, $x_2 \ldots x_n$. Each Element index (x_1, $x_2 \ldots x_n$) is zero based, so the range of x_i is zero to D_i-1, i.e. $x_i = [0, D_i\text{-}1]$. The Element size (E_{Bytes}) of each Element in the Array is denoted as E_S.

1-Dimensional Array

$$\text{Offset}(\text{column}) = \text{Offset}(x_1) = x_1 E_s$$

2-Dimensional Array

$$\text{Offset}(\text{row}, \text{column}) = \text{Offset}(x_1, x_2) = (x_1 D_2 + x_2) E_s$$

3-Dimensional Array

$$\text{Offset}(\text{plane}, \text{row}, \text{column}) = \text{Offset}(x_1, x_2, x_3) = (x_1 D_2 D_3 + x_2 D_3 + x_3) E_s$$

N-Dimensional Array

$$\text{Offset}(x_1, \ldots, \text{plane}, \text{row}, \text{column}) = \text{Offset}(x_1, x_2, \ldots, x_N) = \left(\sum_{i=1}^{N} x_i \left(\prod_{j=i+1}^{N} D_j \right) \right) E_s$$

Appendix C Normalizing Element Lengths

The Multi-Dimensional Array Pack requires all Elements within the Array to be the same length. This can be accomplished by either reducing or inflating selected Elements to the desired length. Reducing data length is possible depending on the application and can be performed if data integrity is not lost during the processes. For example, if the data Elements are null-padded strings then reduce the length by removing trailing null characters, if they do not provide any meaning.

Elements inflated or upsized to match the lengths of larger Elements must follow a consistent method of inflating. Each type of data will require a different method as indicated in Table 2.

Table 2: Data Inflation Methods

Type	Inflation Method
Signed Integer	Add sign extended bytes by adding additional most significant bytes. If the most significant bit of the original value is zero (0), then additional bytes will have all bits zero (0x00). If the most significant bit of the original value is one (1), then additional bytes will have all bits set (0xFF).
Unsigned Integer	Add additional most significant bytes with zero filled bytes (0x00)
32-bit Float	Type Cast 32-bit floating point value into a 64-bit floating point value
Strings	Pad the end of the string with null (0x00) characters

To deflate the data back to the original size, if known, the Data Inflation method is performed in reverse.

Appendix D EPA (Element Processing Algorithm)

An Element Processing Algorithm enables packing or other processing on Array Elements. Table 3 lists the available Element Processing Algorithms.

Table 3: Element Processing Algorithms

EPA Value	Algorithm	Number of Parameters
0x00	Unused	0
0x01	Direct mapping to KLV dictionary - No Element Processing	0
0x02	MISB ST 1201 Floating Point to Integer Mapping - See **Error! Reference source not found.**	2

Appendix D.1 MISB ST 1201 Element Processing

When floating point Elements are formatted into KLV there is the option of using MISB ST 1201 [2] to map floating point values to integers to reduce the number of bytes per Element.

There are two methods specified in MISB ST 1201 for mapping the values: IMAPA and IMAPB. IMAPA uses a minimum, maximum and a precision value to compute the length of a mapped integer value; IMAPB uses a minimum, maximum and a pre-computed length. Table 4 describes the minimum and maximum values per MISB ST 1201.

Table 4: MISB ST 1201 Minimum/Maximum Values

Name	Description
ST 1201 Minimum Value	The Minimum Value used for mapping and reverse mapping any value in the Array to its corresponding packed value - see MISB ST 1201 for further details.
ST 1201 Maximum Value	The Maximum Value used for mapping and reverse mapping any value in the Array to its corresponding packed value - see MISB ST 1201 for further details.

Requirement	
ST 1303-17	When using EPA 0x02, the EPAS value shall contain the Minimum and Maximum values as defined in MISB ST 1303 Table 4: MISB ST 1201 Minimum/Maximum Values.

The value of E_{Bytes} along with the Minimum and Maximum values form the IMAPB mapping parameters for each value in the Array: IMAPB(Minimum, Maximum, E_{Bytes}). Because the length of an Element is already specified in the Array support information, this standard uses the IMAPB method of MISB ST 1201.

Requirement(s)	
ST 1303-18	If EPA is 0x02, then the Array Elements shall be encoded as IMAPB(Minimum, Maximum, E_{Bytes}).
ST 1303-19	The IMAPB Minimum and Maximum parameters shall both be the same size: either both 32-bit or both 64-bit IEEE [6] floating point numbers.

The combined length of the Minimum and Maximum parameters is computed from the total length of the Pack minus the length of the other parameters (N_{Dim}, Dim_1, …, Dim_N, E_{Bytes} EPA and Array of Elements), as shown below.

MinMax Length = Pack Length - ($L(N_{Dim}) + L(Dim_1) + \ldots + L(Dim_N) + L(E_{Bytes}) + L(EPA) + L(\text{Array of Elements})$)

Where:

MinMax Length = L(Minimum) + L(Maximum)

L(**x**) is the length of value **x** in bytes.

MinMax Length will be either 8 bytes or 16 bytes. When the MinMax length is 8 bytes then the Minimum and Maximum values are single precision, so the first four bytes are the Minimum parameter and the last four bytes are the Maximum parameter. When the MinMax length is 16 bytes the Minimum and Maximum values are double precision, so the first 8 bytes are the Minimum parameter and the last 8 bytes are the Maximum parameter.